第1章

微型计算机系统基本组成原理

1.1 重点内容提要

1. 微型计算机系统的硬件结构

从硬件结构看，微型计算机系统基本上采用的是计算机的经典结构——冯·诺依曼结构，即微型计算机系统由运算器、控制器、存储器、输入设备和输出设备五大部分组成；它的数据和程序均以二进制代码形式不加区别地存放在存储器中，存放位置由地址（也是二进制码）指定；其控制器按**指令流驱动**的原理工作，即根据存放在存储器中的指令序列（即程序）工作，并由程序计数器（即指令地址计数器）控制指令的执行，且控制器具有判断能力，能根据计算结果选择不同的动作流程。

上述各部分硬件又是通过地址总线（AB）、数据总线（DB）和控制总线（CB）三大总线联系到一起的，故称为三总线结构。由于使用总线，使硬件各模块之间的相互依赖关系变为模块与总线间的单向依赖关系，即满足相同总线规范的模块就可应用于系统，从而使微型计算机的系统构造比较简单，且具有更大的灵活性和更好的可扩充性、可维修性。根据总线组织方法不同，又可分为单总线、双总线、双重/多重总线三类。

2. 微型计算机的主要组成部分及功能

微型计算机主要由微处理器（MPU）、存储器、I/O 接口和总线四部分组成。

1）微处理器

微处理器是微型计算机的运算和指挥控制中心。不同型号的微型计算机，其性能的差别首先在于其微处理器性能的不同，而微处理器性能又与它的内部结构、硬件配置有关。但无论哪种微处理器，其

内部基本结构总是相同的，都由运算器、控制器和寄存器组三大部分，外加内部总线及缓冲器组成，每个部分又各由一些基本部件组成。

运算器主要由算术逻辑单元 ALU、累加器 ACC、累加锁存器、暂存器和标志寄存器 FR 等部件组成。其主要功能是完成各种算术运算和逻辑运算。

控制器主要由程序计数器 PC、指令寄存器 IR、指令译码器 ID、操作控制器 OC、堆栈指针 SP 和一组通用寄存器等部件组成，是整个微处理器的指挥控制中心。控制器的主要功能是：

- 按照程序逻辑要求，控制程序中指令的执行顺序；
- 根据指令寄存器中的指令码控制每一条指令的执行过程。

寄存器组实质上是微处理器的内部 RAM，可分为专用寄存器和通用寄存器两类。**专用寄存器**（含堆栈指针 SP、程序计数器 PC、标志寄存器 FR 等）的作用是固定的；**通用寄存器**可由程序员规定其用途，一般用于暂时存放需重复使用的某些操作数或中间结果。

对其中一些主要组成部件的功能与工作特点要重点掌握：

（1）算术逻辑单元 ALU 和累加器 ACC

ALU 是运算器的核心，是一种以全加器为核心的具有多种运算功能的组合逻辑电路。用于完成各种算术、逻辑及移位运算。累加器 ACC 提供送入 ALU 的两个运算操作数之一，并存放运算后的结果。而操作结果的某些重要状态或特征则存于标志寄存器 FR，有些机器的 FR 中还存放控制处理器工作方式的控制标志和系统标志。

（2）程序计数器 PC

程序计数器 PC 是维持微处理器有序地执行程序的关键性的、不可缺少的寄存器。它存放着下一条要执行指令的存放地址。并能自动修改指向下一指令的存放地址。

（3）指令寄存器 IR、指令译码器 ID 和操作控制器 OC

指令寄存器 IR 存放从存储器中取出的各条指令的操作码。指令译码器 ID 对 IR 中存放的指令进行译码（分析），确定应该进行什么操作。操作控制器 OC 则依据指令译码器 ID 和时序电路的输出信号，产生执行指令所需的全部微操作控制信号，以控制计算机的各部件执行该指令所规定的操作。由于每条指令所执行的具体操作不同，所以每条指令都有一组不同的控制信号的组合，以确定相应的微操作序列。

（4）堆栈和堆栈指针 SP

堆栈是计算机中一种先进后出的数据结构，由栈区和堆栈指针 SP 组成。栈区用来存放数据，既可由硬件寄存器组成，称为硬件堆栈；又可由 RAM 存储区构成，称为软件堆栈。堆栈指针是用来指示栈顶地址的寄存器，用于自动管理栈区，指示当前数据存入或取出的位置。

堆栈有两种操作：压栈（进栈）和弹栈（出栈），均只能在栈顶进行。堆栈作为数据的一种暂存结构，一般主要用于中断处理和过程调用。

2）存储器

存储器又叫内存或主存，是微型计算机的存储和记忆部件，用以存放数据和程序。

内存是由一个个内存单元组成的，每个内存单元有两个属性：一是它对应有一个地址编码（二进制编码）；二是它所存放的内容，也是二进制编码。二者是完全不同的概念。内存单元的总数目叫做**内存容量**。

微机通过给每个内存单元规定不同地址编码来管理内存。CPU 对内存的操作分为读、写两种：从内存单元取数到 CPU 内部叫做**读操作**；从 CPU 内部送数到内存单元叫做**写操作**。进一步，内存又分为只读存储器(ROM)和随机读写存储器(RAM)两类。

3) 输入输出(I/O)设备的接口

I/O 设备是微机系统的必不可少的组成部分。与 CPU 相比，I/O 设备的工作速度较低，处理的信息格式和逻辑时序往往也不能与 CPU 直接兼容。因此微型机与 I/O 设备间的连接与信息交换不能直接进行，而必须设计一个“接口电路”作为两者间的桥梁。I/O 接口电路也称作 I/O 适配器。

3. 计算机中数的表示方法

计算机中，无论数值还是数的符号，都只能用 0 和 1 来表示。通常专门用一个数的最高位作为符号位：0 表示正数，1 表示负数。这种在计算机中使用的、连同符号位一起数字化了的数，称为机器数。机器数可有不同的表示方法。

1) 有符号数的机器数表示方法

对有符号数，机器数常用的表示方法有原码、反码和补码三种。

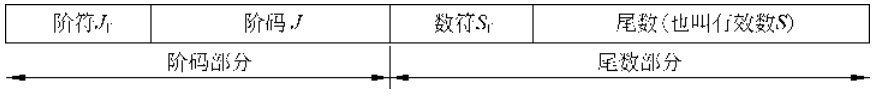
三种码制的最高位都是表示符号位。符号位为 0 时，表示真值为正数，其余位为真值。符号位为 1 时，表示真值为负数，其余位除原码外不再是真值：对于反码，需按位取反才是真值，对于补码，需按位取反加 1 才是真值。即对于正数有：$[X]_{原}=[X]_{反}=[X]_{补}$；而对于负数，三种编码互不相同。所以原码、反码、补码本质上是用来解决负数在机器中表示的三种不同的编码方法。

2) 数的定点和浮点表示

计算机中并不用某个二进制位来表示小数点，而是隐含规定小数点的位置。根据小数点的位置是否固定，数的表示方法可分为定点表示和浮点表示，相应的机器数就叫定点数和浮点数。

所谓定点表示，是指在计算机中所有数的小数点的位置人为约定不变。根据小数点约定位置的不同，定点数有两种形式：定点(纯)整数和定点(纯)小数。两者在计算机中的表示形式没有什么区别，均可用前述的原码、反码和补码表示，其小数点完全靠事先约定而隐含在不同的位置。定点整数约定小数点的位置固定在最低数值位之后；而定点小数约定小数点的位置固定在符号位之后。

当要处理的数是既有整数又有小数的混合小数时，一般采用浮点格式表示。此时，小数点位置不固定。浮点表示的一般格式如下：



其中阶码一般用补码定点整数表示，尾数一般用补码或原码定点小数表示。通常，为保证不损失有效数字，一般应对尾数进行规格化处理，即保证尾数的最高数值位为 1，实际大小通过阶码进行调整。

3) 无符号数的机器数表示方法

无符号数在计算机中通常也有三种表示方法，即位数不等的二进制码(一般有 8 位字

节、16 位字和 32 位双字等)、BCD 码和 ASCII 码。BCD 码又分为压缩 BCD 码和非压缩 BCD 码。

4. 计算机的运算

计算机进行的运算都是有模的运算。模是计量器的最大容量。一个 n 位寄存器能够存放 2^n 个数,它的模为 2^n。对一个 n 位的运算器,当运算结果大于等于 2^n 时,其超出部分被运算器自动丢弃。

1) 补码运算及溢出判别

补码的加减法运算规则如下:

$$[X+Y]_{补} = [X]_{补} + [Y]_{补} \quad (\text{mod } 2^n)$$

$$[X-Y]_{补} = [X]_{补} - [Y]_{补} \quad (\text{mod } 2^n)$$

$$[X-Y]_{补} = [X]_{补} + [-Y]_{补} \quad (\text{mod } 2^n)$$

$$[-Y]_{补} = [Y]_{补} \text{ 连同符号位一起求反加 1(称为}\textbf{求补}\text{)}$$

其中 X、Y 为正、负数均可。上述运算规则说明,补码减法既可用减法,也可用补码加法实现,区别在于:加法产生的进位与减法产生的借位相反,即补码加法有进位,则表明对应补码减法无借位,反之,补码加法无进位,则表明对应补码减法有借位。

在计算机系统中,用补码加法代替补码减法,可省去减法器,从而简化硬件。

对补码运算,当真值满足:$-2^{n-1} \leqslant X, Y, (X \pm Y) < 2^{n-1}$ 时,应用上述规则可得到正确的结果。其中 n 为字长,运算以 2^n 为模。此时,最高位为 0,表示结果为正数;最高位为 1,表示结果为负数。当结果超出补码所能表示的数值范围时,会产生错误的结果,这种现象称为“**溢出**”。计算机运算时要避免产生溢出,万一出现溢出,要能判断,并作出相应处理。

微型机中多采用“双进位位”法判别溢出。即两个补码数相加时,根据最高数值位向符号位的进位 C_1 和符号位向进位位的进位 C_2 来判断:

- 当 $C_2C_1=11$ 或 00 时,表示无溢出;
- 当 $C_2C_1=01$ 时,表示正溢出,即两正数相加结果为负;
- 当 $C_2C_1=10$ 时,表示负溢出,即两负数相加结果为正。

2) BCD 码运算及其十进制调整

进行 BCD 码加减法运算时,每组 4 位二进制码表示的十进制数之间应该遵循“逢十进一”和“借一当十”的规则。但是,由于计算机总是将数作为二进制数来处理的,即每 4 位之间总是按“逢十六进一”和“借一当十六”来处理,所以当 BCD 码运算出现进位和借位时,结果将出错。因此,为得到正确的 BCD 码运算结果,必须对二进制运算结果进行调整,使之符合十进制数运算的进位/借位规则。这种调整称为十进制调整。调整规则如下。

十进制加法调整规则:两个 BCD 数相加,若出现本位和大于 9,或虽不大于 9,但向高位产生了进位,则应在本位作加 6 修正。

十进制减法调整规则:两个 BCD 数相减,若出现本位差超过 9,或虽不超过 9,但向高位有借位,则应在本位作减 6 修正。

通常,在计算机中都设有相应的 BCD 码调整指令。

5. 微型计算机的基本工作原理

微型计算机工作的过程本质上就是执行程序的过程。而程序是为解决某一问题而编写在一起的一个指令序列，因此，了解微机工作的原理，关键是要了解指令和指令执行的过程。

(1) 指令是规定计算机执行特定操作的命令。计算机全部指令的集合称为计算机指令系统，它准确地定义了计算机的处理能力。不同型号的计算机有不同的指令系统，从而形成各种型号计算机的特点和相互间的差异。

(2) 任何一条指令都包括两部分：操作码和地址码。操作码指明要完成操作的性质；地址码也叫操作数，用于指明参加上述规定操作的数据存放的地址或操作数。但要注意，其中地址码可以在指令中显式给出，也可以隐式约定。

(3) 微型计算机每执行一条指令都是分成三个阶段进行：取指令、分析指令和执行指令。而程序的执行过程，实际上就是周而复始地完成这三个阶段操作的过程，直至遇到停机指令才结束整个机器的运行。

理解程序的执行过程时，要注意不同指令的这三段操作并非在各种微机中都是串行完成的，除早期的 8 位微机外，各种 16 位、32 位微机都可将这几阶段操作分配给两个或两个以上的独立部件并行完成，形成流水线结构，使不同指令的取指、分析、执行三个阶段可并行处理，从而加速程序的执行过程。

1.2　疑难问题解答

问 1.1　BCD 码与纯二进制数的主要区别是什么？

解：BCD 码是一种用二进制编码表示的十进制数，所以 BCD 码与纯二进制数的主要区别是二者对应的二进制位的**权**不同。以 8 位二进制码 $D_7D_6D_5D_4D_3D_2D_1D_0$ 为例，若看做纯二进制数，各位对应的权从高位到低位分别为：2^7、2^6、2^5、2^4、2^3、2^2、2^1 和 2^0。若看做压缩 BCD 码，各位对应的权则为：$2^3\times10$、$2^2\times10$、$2^1\times10$、$2^0\times10$、2^3、2^2、2^1 和 2^0。

问 1.2　字节和字长这两个概念有何区别？

解：字节指由 8 个二进制位组成的基本数据单元，是与机器无关的概念。字长是指计算机内部一次可以处理的二进制数码的位数，决定于通用寄存器、ALU 的位数和数据总线的宽度等，依赖于具体机器。

字节是计算机中表示存储容量的基本单位，而字长则是衡量微机系统精度和速度的重要指标。字长越长，1 个字所能表示的数据精度就越高；在完成同样精度的运算时，处理速度也越高。例如要完成 1 个 32 位的加法，8 位机需要执行 4 个 8 位加法，16 位机需要执行 2 个 16 位加法，而 32 位机只需执行 1 个 32 位加法。然而，字长越长，计算机的硬件也越复杂。

问 1.3　为什么把微机的基本结构说成是总线结构？

解：微型计算机从硬件结构看是由微处理器、存储器和 I/O 接口三大部分组成，而各部分又是通过总线联系在一起的。正是因为总线，才使各部分(模块)之间的相互依赖关

系变为模块与总线间的单向依赖关系，即只要满足总线规范的模块就可应用于系统，从而使微型计算机的系统构造比较简单，且具有更大的灵活性和更好的可扩充性、可维修性。所以常把微型计算机的基本结构说成是总线结构。

问 1.4 在双重总线结构中，局部总线和全局总线有什么联系？

解：在双重总线结构中，局部总线可以看做 CPU 模块的内部总线，而全局总线则是实现多个 CPU 模块（或其他主控器）连接、组成并行处理系统的外部总线。CPU 访问模块内的局部存储器和 I/O 接口时，无需使用全局总线，即各 CPU 模块并行处理，只有当 CPU 需要访问挂接在全局总线上的全局存储器、全局 I/O 或其他 CPU 模块资源时，才通过总线仲裁器仲裁使用全局总线。

问 1.5 计算机系统中常采用补码形式表示和存储数据，其原因是什么？

解：用补码表示和存储数据的好处：一是补码减法可用加法实现，且对应负数的补码可用求补运算完成，因而在运算器中可省去减法器；二是无符号数的运算可与补码运算共用一个运算器。所以，用补码表示和存储数据可简化运算器的设计。

问 1.6 程序计数器的位数取决于什么？

解：程序计数器存放下一条要执行的指令的地址，而指令是预存在存储器中的，所以它的位数取决于存储器的容量，而存储容量又是由 CPU 地址总线的位数确定的，所以也可以说取决于 CPU 地址总线的位数。

问 1.7 在微型计算机中，程序执行的时间就是程序中各条指令执行时间的总和吗？

解：不一定。在采用流水线结构的微型计算机中，由于不同指令的取指、分析、执行三个阶段可并行处理，从整体上使程序的执行时间减少了，这时，程序执行的时间就不是程序中各条指令执行时间的总和。

问 1.8 微处理器中采用流水线技术后，是否意味着每条指令的执行时间明显缩短了？

解：不能缩短每条指令的执行时间。采用流水线技术后，并没有加速单条指令的执行，每条指令的操作步骤（取指、分析、执行）一个也不能少，只是多条指令的不同操作步骤可由多个独立的功能部件同时执行，因而从总体看加快了指令流速度，缩短了程序执行时间。

问 1.9 有符号数与无符号数的“溢出”判别方法有何不同？

解：位数相同的无符号数和补码表示的有符号数所能表示的数值范围不同，所以“溢出”的判别方法也各不相同。以加/减法为例，对无符号数而言，只要运算产生了进位/借位，就说明运算结果超出了结果单元所能表示的数值范围，即发生了“溢出”；而对补码表示的有符号数而言，其运算是否“溢出”常用“双进位位法”判别。所以，微机中无符号数的“溢出”判别用标志寄存器的进位标志 CF，而有符号数的“溢出”判别则用标志寄存器的溢出标志 OF。

问 1.10 计算机运算是如何区分有符号数和无符号数、组合 BCD 码和非组合 BCD 码、BCD 码和 ASCII 码的？

解：对加法和减法类运算，计算机本身不区分有符号数和无符号数，它们共用一个运算器进行运算，有符号数的符号位就如同数值位一样一起参与运算，但会给出一组相关的重要状态标志，如进位标志 CF、符号标志 SF、溢出标志 OF、零标志 ZF 和半进位标志 AF 等，运算后用户可以根据是有符号数还是无符号数运算，通过条件转移指令选择不同的状态标志来控制程序，如无符号数大小比较用 ZF 和 CF 标志位，而有符号数大小比较则要用 ZF、

OF 和 SF 三个标志位；对乘/除法和移位操作则要根据操作对象是有符号数还是无符号数使用不同的指令来区分，如 80x86 中，有符号数的乘/除法用 IMUL/IDIV 指令、移位用 SAL/SAR 指令，而无符号数的乘/除法则要用 MUL/DIV 指令、移位用 SHL/SHR 指令。

同样，计算机运算并不区分组合 BCD 码、非组合 BCD 码和 ASCII 码，而把它们当二进制数参与运算，遵循二进制运算的法则，但提供一组相关状态标志（如 CF 和 AF）和 BCD 调整指令供编程者调整结果用，所以，BCD 码运算后，通常都要跟一条相应的 BCD 调整指令来调整结果，ASCII 码运算后，使用非压缩 BCD 码调整指令调整，再转换为 ASCII 码。如以下程序将 ASCII 数相加并将结果转换为 ASCII 码：

```
MOV AL,'9'      ; (AL) = 39H
ADD AL,'8'      ; 二进制相加,(AL) = 71H,CF = 0,AF = 1
AAA             ; 低 4 位加 6 调整,高位清 0,(AL) = 07H,CF = AF = 1
OR AL,30H       ; (AL) = 37H = '7'
```

问 1.11 指令流驱动的 CPU 与数据流驱动的 CPU 有何区别？

解：主要在于控制指令执行的方式不同。指令流驱动基于指令流控制工作，即 CPU 是依据程序计数器控制顺序执行存放在存储器中的指令序列（程序）来工作的。而数据流驱动则是基于数据流控制工作，依据数据流分析允许指令按照不同于程序中指定的顺序发送给执行部件执行，即满足执行条件的指令可以优先执行。

问 1.12 如何理解流水线的级数和条数？

解：流水线的级数越多，表明指令执行时重叠的步骤越多，程序总的执行时间越少，但 CPU 硬件也越复杂，当流水线级数达到一定数量，而指令步骤难以分解时，增加流水线级数只会增加 CPU 硬件的复杂性，而对提高程序的运算速度影响将不会太大。

同样，流水线的条数越多，表明 CPU 同时执行的指令数越多，CPU 并行处理能力越强，硬件也越复杂。与流水线级数一样，流水线的条数也是有限的。

1.3 典型例题解析

1. 选择题

例 1.1 关于 CPU，以下_____的说法是错误的。

A. 是中央处理单元的简称　　B. 能高速准确地执行指令

C. 能直接为用户解决各种实际问题　　D. 是计算机的核心部件

解：单纯的 CPU 不是计算机，不能独立工作。因而不能直接为用户解决各种实际问题。

答案为：C

例 1.2 CPU 中 ALU 部件主要完成_____。

A. 地址指针的转换　　B. 中断处理

C. 算术和逻辑运算　　D. 产生各种时序

解：ALU 是算术逻辑运算单元的简称，主要用于完成各种算术和逻辑运算。

答案为：C

例 1.3 根据冯·诺依曼结构，最基本的计算机应用_____单元构成，冯·诺依曼计算机也称为_____计算机。

A. 3　　B. 4　　C. 5　　D. 6

A. 控制流　　B. 指令流　　C. 微代码　　D. 程序

解：冯·诺依曼结构计算机在硬件上由运算器、控制器、存储器、输入设备和输出设备五大部分组成，其控制器按**指令流驱动**的原理工作，所以也称为指令流计算机。

答案为：C、B

例 1.4 一个 8 位二进制整数，若采用补码表示，且由 4 个 1 和 4 个 0 组成，则最小值为_____。

A. −120　　B. −7　　C. −112　　D. −121

解：此题是求最小值，所以必定是负数，符号位为 1。而补码负数的特点**是数值位越小，其绝对值越大，即负得越多，真值越小，反之亦然**。所以，由 4 个 1 和 4 个 0 组成的补码数中，真值最小的补码数为：10000111，即真值为：−121。

答案为：D

例 1.5 8 位补码操作数“10010011”等值扩展为 16 位后，其机器数为_____。

A. 1111111110010011　　B. 0000000010010011

C. 1000000010010011

解：有符号补码数的扩展，是用符号位充填高位，其真值保持不变。但对无符号二进制数而言，各二进制位均是有效数值位，所以无符号二进制数的等值扩展不能用最高数值位充填扩充的高位，而是用 0 充填扩充的高位。

答案为：A

例 1.6 计算机内的“溢出”是指其运算的结果_____。

A. 为无穷大

B. 超出了计算机内存储单元所能存储的数值范围

C. 超出了该指令所指定的结果单元所能存储的数值范围

解：计算机进行的运算都是有模的运算，即对一个 n 位的运算器，当运算结果大于等于 2^n 时，其超出部分被运算器自动丢弃。所以，当运算结果超出了结果单元所能表示的数值范围时，会产生错误的结果，这种现象称为“溢出”。需要说明的是“溢出”与数的表示方法有关，因而“溢出”的判别方法也不相同，以加/减法为例，对补码表示的有符号数常用“双进位位”法判别；而对无符号数，只要运算有进位或借位就表明运算有“溢出”。

答案为：C

例 1.7 下列无符号数中最大的数是_____。

A. $(765)_8$　　B. $(319)_{16}$　　C. $(1010100110)_2$　　D. $(789)_{10}$

解：将答案 A、B 代表的数转换成等值的二进制数，为：

$(765)_8=(111\ 110\ 101)_2$　　$(319)_{16}=(0011\ 0001\ 1001)_2$

即 A、B、C 三个答案中最大的数为：$(319)_{16}=(0011\ 0001\ 1001)_2$

该数所代表的十进制真值为：$(319)_{16}=(3\times16^2+1\times16+9)_{10}=(793)_{10}$，即答案为 B。

答案为：B

例 1.8 计算机中微操作控制电路的实现方式有三种，它们为：_____。

A. 组合逻辑控制、同步控制和异步控制

B. 微程序控制、DMA 控制和中断控制

C. 组合逻辑控制、PLA 控制和微程序控制

D. 组合逻辑控制、PLA 控制和程序控制

解：PLA 是可编程逻辑阵列，通过编程可实现各种功能的组合逻辑控制。所以，除用组合逻辑（硬件控制）和微程序控制（软件控制）外，PLA 也常用于实现计算机中微操作控制电路。

答案为：C

例 1.9　某计算机字长为 16 位，其浮点数格式为：阶符 1 位，阶码 5 位（补码表示），数符 1 位，尾数 9 位（原码表示），则二进制数 $X=-0.00110101$ 的规格化浮点数为_____。

A. 111110 1110101000　　B. 111110 0110101000

C. 111111 1011010100

解：$X=-0.00110101=-0.110101\times2^{-2}$

将阶码用 6 位补码表示，尾数用 10 位原码表示，其规格化浮点数为：

1	11110	1	110101000

答案为：A

例 1.10　微处理器的字长、主频、ALU 结构以及_____等功能是影响其处理速度的主要因素。

A. 有无中断处理　　B. 是否采用微程序控制

C. 有无 DMA 功能　　D. 有无 cache 存储器

解：cache 存储器常采用高速 SRAM，存取速度快，在 CPU 与内存间设置这样的存储器可以保证 CPU 以尽可能快的速度与内存打交道。

答案为：D

2. 判断题

例 1.11　任何一条计算机指令都显式包含操作码和操作数两部分。

解：计算机指令由操作码和操作数两部分组成，其中操作码需显式给出，但操作数可以是显式的也可以是隐含的。如 80x86 的查表指令 XLAT、BCD 加法调整指令 DAA 均没有显式给出操作数，而是使用隐含的操作数（BX、AL）。

答案为：说法不正确。

例 1.12　采用流水线技术后，有利于简化计算机的硬件设计，加速指令的执行过程。

解：采用流水线技术，从整体上加速了指令流的执行过程，但它是以增加硬件的复杂性为代价的，有多少级流水线，就至少要有多少个独立功能部件支持，因而不能简化计算机的硬件设计。另外，对单条指令的执行过程也不能加速。

答案为：说法不正确。

例 1.13　在 CPU 中执行的算术和逻辑运算都是按位进行且各位之间是独立无关的。

解：逻辑运算是按位进行且各位之间是独立无关的，但算术运算不是按位进行的，如加/减法运算，相邻位之间可能产生进位/借位。

答案为：说法不正确。

例 1.14 任何微处理器都具有运算和控制功能，但不具备存储功能。

解：运算和控制功能是任何微处理器必不可少的功能。存储功能一般也是微处理器的功能之一，但不是必备的功能，如单片机有存储功能，高档微机 80486、80586 也含一定数量的 cache 存储器，具有存储功能，微处理器中的寄存器从广义上说也属存储部件。

答案为：说法不正确。

3. 计算题

例 1.15 若 $X=-63, Y=+127$，求 $[X-Y]_{补}$。要求写出运算过程，并指明运算后借位标志、符号标志及溢出标志的情况。

解：此题关键是掌握好 $[Y]_{补}$、$[-Y]_{补}$ 的求法和溢出的“双进位位”判别法。

由题可得：$[X]_{补}=[-63]_{补}=11000001$

$$[Y]_{补}=01111111,\quad [-Y]_{补}=10000001$$

按补码运算规则有：$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$，即：

$$\begin{array}{r} [X]_{补}=11000001 \\ +[-Y]_{补}=10000001 \\ \hline \boxed{1}\ 01000010=42\mathrm{H} \end{array}$$

即 $[X-Y]_{补}=42\mathrm{H}$。

符号位向进位位的进位 $C_2=1$，最高数据位向符号位的进位 $C_1=0$，由“双进位位”判别法得：溢出标志 $OF=C_1 \oplus C_2=1$，表明结果溢出。

符号位为 0，所示符号标志 SF=0。

补码加法有进位，表明补码减法无借位，所示借位标志 CF=0。

例 1.16 (1) 设 $[X]_{补}=10101010\mathrm{B}$，则 $\left[-\frac{1}{2}X\right]_{补}$ 的值是什么？

(2) 设 $[X]_{补}=11010100\mathrm{B}$，则 $\left[-\frac{1}{4}X\right]_{补}$ 的值是什么？

解：二进制数的倍增和倍减可用移位操作实现，关键是要区分有符号数和无符号数的移位方法。有符号数用算术移位，无符号数用逻辑移位。两者左移含义相同，即数值位(含符号位)依次左移，用 0 充填最低位。右移时，数值位(含符号位)依次右移，但最高位充填方法不同，算术右移用原符号位充填，而逻辑右移用 0 充填。

此题是有符号数倍减，用算术右移，于是：

(1) $\left[\frac{1}{2}X\right]_{补}=[X]_{补}$ 算术右移 1 位=(10101010)算术右移 1 位=11010101B。

(2) $\left[\frac{1}{4}X\right]_{补}=[X]_{补}$ 算术右移 2 位=(11010100)算术右移 2 位=11110101B，

所以 $\left[-\frac{1}{4}X\right]_{补}=\left[\frac{1}{4}X\right]_{补}$ 连同符号位一起求反加 1=00001011B。

例 1.17 设浮点数的表示格式为：阶码 4 位(包括阶符 1 位)、尾数 8 位(包括尾符 1

位)。阶码和尾数均用补码表示。写出二进制数 $X=-0.0110101$ 的规格化浮点数表示。

解：首先把 X 写成规格化的浮点真值数：$X=-0.1101010\times2^{-1}$，则规格化的浮点补码数如下：

1	111	1	0010110
阶符	阶码	尾符	尾数

例 1.18 设阶码用原码表示，尾数用补码表示，求下列浮点机器数的真值。

1	0010	1	0010011001
阶符	阶码	尾符	尾数

解：阶码真值为：-2，尾数真值为：-0.1101100111，所以

$$浮点数真值=-0.1101100111\times2^{-2}=-0.001101100111$$

例 1.19 十进制数 1943，其对应的十六进制数是_____ H，压缩 BCD 数是_____ H，非压缩 BCD 数是_____ H。

解：十进制数 1943 可用连续除 16 取余的方法转换成 16 进制数，结果为 797H。

转换成压缩 BCD 码的方法是每位十进制数用 4 位二进制数进行编码，二进制编码形式为：

0001 1001 0100 0011

写成十六进制形式为 1943H。

而非压缩 BCD 是每位十进制数用 8 位二进制数进行编码，二进制编码形式为：

00000001 00001001 00000100 00000011

写成十六进制形式为：01090403H。

1.4 教材习题选解

1.5 试填写出表 1.1 中各数对应的 8 位原码、反码和补码。

表 1.1

十进制数	原　码	反　码	补　码
+0			
+15			
+62			
+127			
−0			
−18			
−10			
−100			
−127			
−128			

解：各数对应的 8 位原码、反码和补码如表 1.2 所示。

表 1.2

十 进 制 数	原　　码	反　　码	补　　码
＋0	00000000	00000000	00000000
＋15	00001111	00001111	00001111
＋62	00111110	00111110	00111110
＋127	01111111	01111111	01111111
－0	10000000	11111111	00000000
－10	10001010	11110101	11110110
－18	10010010	11101101	11101110
－100	11100100	10011011	10011100
－127	11111111	10000000	10000001
－128	无	无	10000000

1.7　填写表 1.3 中各机器码分别为原码、反码、补码、无符号二进制码和压缩 BCD 码时所对应的十进制真值。

表 1.3

机　器　码	对应十进制真值				
	原　　码	反　　码	补　　码	无符号二进制码	压缩 BCD 码
10001001					
10000000					
10010110					
01010011					
00000111					
11111111					

解：各机器码分别为原码、反码、补码、无符号二进制码和压缩 BCD 码时，所对应的十进制真值如表 1.4 所示。

表 1.4

机　器　码	对应十进制真值				
	原　　码	反　　码	补　　码	无符号二进制码	压缩 BCD 码
10001001	－9	－118	－119	137	89
10000000	－0	－127	－128	128	80
10010110	－22	－105	－106	150	96
01010011	83	83	83	83	53
00000111	7	7	7	7	7
11111111	－127	－0	－1	255	无效

1.9　已知某微型机的浮点数采用 IEEE 单精度浮点数格式。请问：

(1) 该浮点数所能表示的真值范围是多少？

(2) 浮点数 FB8789ABH 和 58EBA987H 代表的十进制数分别是多少？

(3) $(-86.57)_{10}$对应的机器数(浮点数)是什么?

解:(1) 该格式浮点数所能表示的最小数对应的机器数为(阶码最大,有效数最小):

1	1111 1110	1111 1111 1111 1111 1111 111

其对应的真值为:　　$-(2-2^{-23})\times 2^{127}$

该格式浮点数所能表示的最大数对应的机器数为(阶码最大,有效数最大):

0	1111 1110	1111 1111 1111 1111 1111 111

该机器数代表的真值为:　　$(2-2^{-23})\times 2^{127}$

由此,可确定该浮点数所能表示的真值范围为:$-(2-2^{-23})\times 2^{127}\sim(2-2^{-23})\times 2^{127}$。

(2) 浮点数 FB8789ABH 的 IEEE 单精度格式为:

1	1111 0111	000 0111 1000 1001 1010 1011

阶码为:　　$(11110111)_2-(01111111)_2=120$

十进制真值为:　$(-1.000\ 0111\ 1000\ 1001\ 1010\ 1011)_2\times 2^{120}$

$=(-1000\ 0111\ 1000\ 1001\ 1010\ 1011)_2\times 2^{97}$

$=-8882603\times 2^{97}$

浮点数 58EBA987H 的 IEEE 单精度格式为:

0	1011 0001	110 1011 1010 1001 1000 0111

阶码为:　　$(10110001)_2-(01111111)_2=50$

十进制真值为:　$(1.110\ 1011\ 1010\ 1001\ 1000\ 0111)_2\times 2^{50}$

$=(1110\ 1011\ 1010\ 1001\ 1000\ 0111)_2\times 2^{27}$

$=15444359\times 2^{27}$

(3) $(-86.57)_{10}=(-1010110.100100011)_2=(-1.010110100100011)_2\times 2^6$

机器数表示的阶码为:127+6=1000 0101B,所以该机器数为:

1	1000 0101	0101 1010 0100 0110 0000 000

1.17　请自行写出主教材 1.3.3 小节中给出的程序的详细执行步骤。

解:1)“MOV A,5CH”详细执行步骤

(1) 将 PC 内容 1000H 送地址寄存器 MAR。

(2) PC 值自动加 1,为取下一个字节机器码作准备。

(3) MAR 中内容经地址译码器译码,找到内存储器 1000H 单元。

(4) CPU 发出读命令。

(5) 将 1000H 单元内容 B0H 读出,送至数据寄存器 MDR。

(6) 由于 B0H 是操作码,故将它从 MDR 中经内部总线送至指令寄存器 IR。

(7) 经指令译码器 ID 译码,由操作控制器 OC 发出相应于操作码的控制信号。

以下取操作数。

(8) 将 PC 内容 1001H 送至 MAR。

(9) PC 值自动加 1。

(10) MAR 中内容经地址译码器译码,找到 1001H 存储单元。

(11) CPU 发出读命令。

(12) 将 1001H 单元内容 5CH 读至 MDR。

以下执行指令。

(13) 因 5CH 是操作数,将它经内部总线送至操作码规定好的累加器 A。

2) "ADD A,2EH" 详细执行步骤

(1) 将 PC 内容 1002H 送至地址寄存器 MAR。

(2) PC 值自动加 1,为取下一个字节机器码作准备。

(3) MAR 中内容经地址译码器译码,找到内存储器 1002H 单元。

(4) CPU 发出读命令。

(5) 将 1002H 单元内容 04H 读出,送至数据寄存器 MDR。

(6) 由于 04H 是操作码,故将它从 MDR 中经内部总线送至指令寄存器 IR。

(7) 经指令译码器 ID 译码,由操作控制器 OC 发出相应于操作码的控制信号。

(8) 将 PC 内容 1003H 送 MAR。

(9) PC 值自动加 1。

(10) MAR 中内容经地址译码器译码,找到 1003H 存储单元。

(11) CPU 发出读命令。

(12) 将 1003H 单元内容 2EH 读至 MDR。

(13) 因 2EH 是操作数,将它经内部总线送至 ALU。

(14) 执行: 2EH+5CH 送 A 累加器,修改标志寄存器。

3) "JO 100AH"详细执行步骤

(1) 将 PC 内容 1004H 送至地址寄存器 MAR。

(2) PC 值自动加 1,为取下一个字节机器码作准备。

(3) MAR 中内容经地址译码器译码,找到内存储器 1004H 单元。

(4) CPU 发出读命令。

(5) 将 1004H 单元内容 70H 读出,送至数据寄存器 MDR。

(6) 由于 70H 是操作码,故将它从 MDR 中经内部总线送至指令寄存器 IR。

(7) 经指令译码器 ID 译码,由操作控制器 OC 发出相应于操作码的控制信号。

以下取操作数。

(8) 将 PC 内容 1005H 送至 MAR。

(9) PC 值自动加 1。

(10) MAR 中内容经地址译码器译码,找到 1005H 存储单元。

(11) CPU 发出读命令。

(12) 将 1005H 单元内容 0AH 读至 MDR。

(13) 因 0AH 是操作数,将它经内部总线暂存作为低 8 位地址。

(14) 将 PC 内容 1006H 送 MAR。

(15) PC 值自动加 1。

(16) MAR 中内容经地址译码器译码,找到 1006H 存储单元。

(17) CPU 发出读命令。

(18) 将 1006H 单元内容 10H 读至 MDR。

(19) 因 10H 是操作数,将它与暂存的低 8 位地址结合形成 16 位地址。

以下执行指令。

(20) 因加法(5CH+2EH=8AH)有溢出,将 100A 送 PC。

(下一步将跳过下一条指令,执行 100AH 单元指令)

4) "MOV (0200H),A"详细执行步骤(此指令因加法溢出不执行)

(1) 将 PC 内容 1007H 送至地址寄存器 MAR。

(2) PC 值自动加 1,为取下一个字节机器码作准备。

(3) MAR 中内容经地址译码器译码,找到内存储器 1007H 单元。

(4) CPU 发出读命令。

(5) 将 1007H 单元内容 A2H 读出,送至数据寄存器 MDR。

(6) 由于 A2H 是操作码,故将它从 MDR 中经内部总线送至指令寄存器 IR。

(7) 经指令译码器 ID 译码,由操作控制器 OC 发出相应于操作码的控制信号。

以下取操作数。

(8) 将 PC 内容 1008H 送至 MAR。

(9) PC 值自动加 1。

(10) MAR 中内容经地址译码器译码,找到 1008H 存储单元。

(11) CPU 发出读命令。

(12) 将 1008H 单元内容 00H 读至 MDR。

(13) 因 00H 是操作数,将它经内部总线暂存作为低 8 位地址。

(14) 将 PC 内容 1009H 送至 MAR。

(15) PC 值自动加 1。

(16) MAR 中内容经地址译码器译码,找到 1009H 存储单元。

(17) CPU 发出读命令。

(18) 将 1009H 单元内容 02H 读至 MDR。

(19) 因 02H 是操作数,将它与暂存的低 8 位地址结合形成 16 位地址 0200H 送至 MAR。

以下执行指令。

(20) CPU 发出写命令。

(21) 将 A 内容送 0200H 单元。

5)“HLT”详细执行步骤

(1) 将 PC 内容 100AH 送至地址寄存器 MAR。

(2) PC 值自动加 1,为取下一个字节机器码作准备。

(3) MAR 中内容经地址译码器译码,找到内存储器 100AH 单元。

(4) CPU 发出读命令。

(5) 将 100AH 单元内容 F4H 读出,送至数据寄存器 MDR。

(6) 由于 F4H 是操作码,故将它从 MDR 中经内部总线送至指令寄存器 IR。

(7) 经指令译码器 ID 译码,由操作控制器 OC 发出相应于操作码的控制信号。

(8) 停机。

第2章

微处理器和指令系统

2.1 重点内容提要

1. 8086/8088 的指令流水线和存储器分段

1）指令流水线

8086/8088 CPU 在结构上由执行单元 EU 和总线接口单元 BIU 两个独立的功能部件组成。BIU 中设置的指令预取队列使 EU 和 BIU 可以并行工作，即取指令操作和分析、执行指令操作重叠进行，从而形成了两级“指令流水线”结构。这种“流水线”技术的引入，减少了 CPU 为取指令而必须等待的时间，提高了 CPU 的利用率，加快了整机的运行速度，另外也降低了 CPU 对存储器存取速度的要求。

2）存储器分段管理机制

8086/8088 的内部寄存器只有 16 位，这就是说它能处理的地址信息仅 16 位，即最大寻址空间只有 64KB。为达到寻址 1MB 存储空间的目的，8086/8088 采用了一种巧妙的存储器分段方法，即将 1MB 的物理存储空间分成若干逻辑段，每个逻辑段的最大长度为 64KB。

这种逻辑段的划分可以连续、分离、部分重叠或完全重叠。这主要取决对各个段寄存器(CS、SS、DS 和 ES)的预置内容。

采用存储器分段管理后，用户编程使用逻辑地址，由段基址和段内偏移地址两部分组成，两者都是 16 位，段基址由段寄存器指示。CPU 访问存储器时，地址总线 AB 上送出的是物理地址(20 位地址码)。将编程使用的逻辑地址转换成物理地址是由 BIU 中的地址加法器完成的，变换关系为：

$$\text{物理地址} = \text{段寄存器} \times 16 + \text{偏移地址}$$

2. Pentium 微处理器

Pentium 在结构上主要由执行单元、指令 cache、数据 cache、指令

预取单元、指令译码单元、地址转换与管理单元、总线单元以及控制器等组成，其中核心是执行单元。在 Pentium 的设计中，采用了超标量体系结构、独立的指令 cache 和数据 cache 以及分支指令预测等先进技术。它的外部数据总线为 64 位，但芯片内部 ALU 和通用寄存器仍是 32 位，所以 Pentium 是 32 位微处理器。

1) Pentium 的编程结构

Pentium 包含一组内部寄存器供编程使用。这组寄存器又分为基本寄存器、系统寄存器、调试与测试寄存器(模型专用寄存器)和浮点寄存器。基本寄存器是编程者要求重点理解和熟练掌握的，这些寄存器如图 2.1 所示。

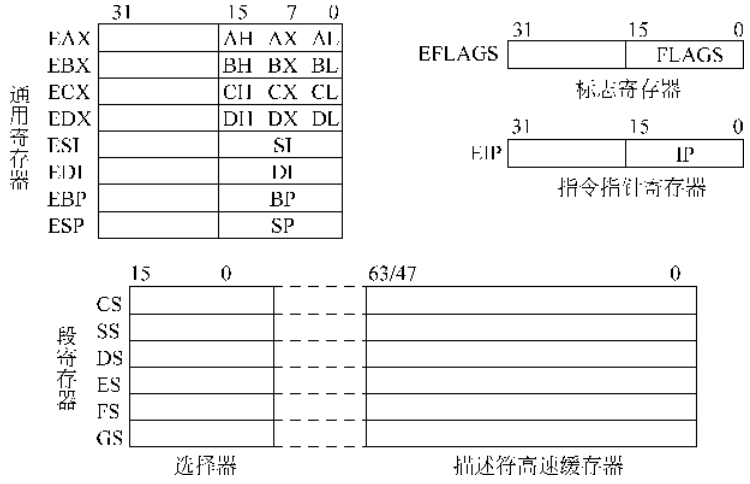


图 2.1　基本寄存器

(1) 通用寄存器

8 个 32 位通用寄存器 EAX、EBX、ECX、EDX、ESI、EDI、EBP、ESP 是在 8086/8088 的 8 个 16 位寄存器基础上扩展位数而来的。为了与 8086/8088 兼容，它们的低 16 位可以单独访问，并以同 8086/8088 中相同的名称命名：AX、BX、CX、DX、SI、DI、BP、SP。其中 AX、BX、CX、DX 还可进一步分成两个 8 位寄存器单独访问，且同样有自己独立的名称：AH、AL，BH、BL，CH、CL，DH、DL。

这些寄存器通常也有自己特殊的用法：(E)SP 用于指示栈顶指针，称为**堆栈寄存器**；BX 和 BP 在 16 位寻址时用作**基址寄存器**；SI 和 DI 在 16 位寻址时用作**变址寄存器**；(E)CX 在循环指令和串操作的重复前缀中用作循环次数或重复次数寄存器，也称为**循环计数寄存器**；DX 的特殊用法是在 I/O 指令间接寻址中作**端口地址寄存器**；(E)AX 用作**累加器**，所有的 I/O 指令及一部分串操作必须使用 EAX、AX 或 AL 来执行，另外还有一些指令使用 EAX、AX 及由 AX 分出的 AL、AH 作为缺省的操作数，如乘法、除法指令。

(2) 指令指针寄存器(EIP)

EIP 用于保存下一条待预取指令相对于代码段基址(由 CS 提供)的偏移量。它的低 16 位也可以单独访问，并称之为 IP 寄存器。当 80x86/Pentium 工作在 32 位操作方式时，采用 32 位的 EIP；工作在 16 位操作方式，采用 16 位的 IP。

(3) 标志寄存器(EFLAGS)

标志寄存器 EFLAGS 是 32 位的,它是在 8086/8088/80286 标志寄存器 FLAGS 的基础上扩充而来的,共定义了三类 17 种(18 位)标志。

(4) 段寄存器

6 个段寄存器用于决定程序使用的存储器区域块。它们的功能是不同的,CS 指明当前的代码段; SS 指明当前的堆栈段; DS、ES、FS 和 GS 指明当前的四个数据段。

2) Pentium 的四种工作方式

Pentium 有四种工作方式:实地址方式、保护虚拟地址方式、虚拟 8086 方式和系统管理方式。

实地址方式是在加电或复位后自动建立起的一种工作方式。在这种工作方式下,Pentium 的工作原理与 8086 方式相同,又称为 8086 方式。保护虚拟地址方式是一种建立在虚拟存储器和保护机制基础上的工作方式,CPU 可访问的物理存储器空间为 4GB,虚拟存储器空间为 64TB。虚拟 8086 方式实际上就是运行在保护环境中的 8086 方式,既支持保护机制又支持分页式内存管理,并可进行任务切换,但同时又与 8086 兼容,内存寻址空间为 1MB,程序中指定的逻辑地址按 8086 方式解释。

系统管理方式 SMM 用于实现高级管理功能,如对电源管理以及为操作系统和正在运行的程序提供安全性保护。

3) 保护方式下的存储器分段管理

保护方式下的存储器分段管理与 8086 方式有所不同。要理解这种分段管理,必须搞清楚段的种类、段描述符、描述符表、描述符的选择符,以及系统地址寄存器。段有代码段、堆栈段、数据段和系统段(LDT、IDT、TSS)之分。每个段用一个段描述符定义,内容包括段的基址、属性和长度。描述符表用于存放逻辑段的段描述符。段寄存器存放的不再是直接的段基址,而是指向某个段描述符的 16 位的选择符。

Pentium 有两种描述符表:全局描述符表 GDT 和局部描述符表 LDT。全局描述符表 GDT 是 80x86/Pentium 用来定义全局存储器地址空间的一种机制,该表存放着操作系统使用的和任务公用的段描述符,这些描述符标识全局存储器中的段。局部描述符表 LDT 则是每个任务用来定义局部存储器地址空间的一种机制,LDT 中的段描述符可用来访问当前任务的存储器段中代码和数据。

当将一个选择符装入一个段选择器时,处理器将自动从 GDT 或 LDT 中找到其对应的段描述符装入相应段寄存器的描述符高速缓存器中。以后,每当访问存储器时,与所用段相关的段描述符高速缓冲器就自动参与该次存储器访问操作。段基地址成为线性地址或物理地址计算中的一个分量,界限用于段限检查操作,属性则对照所要求的存储器访问类型进行检查。线性地址的生成方法如下:

线性地址=段描述符高速缓存器中段基址+偏移地址

不使用页部件时,线性地址即为物理地址; 使用页部件时,上述线性地址需经页管理部件使用页目录和页表转换成物理地址。

3. 80x86/Pentium 的数据类型与寻址方式

1) 数据类型

80x86/Pentium 支持无符号二进制数、带符号的二进制定点整数、浮点数(486 以上

CPU)、BCD 数、串数据、ASCII 码数据和指针数据共 7 类数据。这些数据有些仅 CPU 支持,有些仅 FPU 支持,也有些是两者都支持的,但最基本的仍为字节、字和双字数据。需要记住,**80x86 多字节数据的存放原则是低位字节在低端地址,高位字节在高端地址,而低位字节的地址是多字节数据的访问地址**。

2) 寻址方式

寻址方式是指寻找指令中操作数地址的方式。一般微机系统都支持**立即数寻址**、**寄存器寻址**和**存储器寻址**三类寻址方式,只是不同 CPU 的存储器寻址方式形式可能有所不同。

在 80x86/Pentium 系列 MPU 中,任何内存实际地址(PA)都由两部分组成,即内存单元所在段的基址和此单元与段基址的距离——段内偏移地址(也叫偏移量)。为适应处理不同数据结构的需要,大多数情况下,在指令中并不直接给出操作数的地址,而是给出构成操作数地址的地址分量以及如何计算操作数地址的方法。为此,80x86/Pentium 将段内偏移地址分解成四个基本部分(称为偏移地址四元素):基址寄存器内容、变址寄存器内容、比例因子和位移量。由这四元素组合形成的偏移地址(称为有效地址 EA)与四元素的关系如下:

EA=基址+(变址×比例因子)+位移量

根据四元素在该式中的取舍不同,Pentium 可组合出 9 种不同的存储器寻址方式,如表 2.1 所示。

表 2.1　80x86/Pentium 存储器寻址的 9 种方式

指令寻址方式	有效地址的计算方法	备　注
直接寻址	EA=指令操作数部分直接给出的地址码	16 位和 32 位寻址
寄存器间接寻址	EA=[间接寄存器]	16 位和 32 位寻址
基址寻址	EA=[基址寄存器]+位移量	16 位和 32 位寻址
变址寻址	EA=[变址寄存器]+位移量	16 位和 32 位寻址
比例变址寻址	EA=[变址寄存器]×比例因子+位移量	32 位寻址
基址加变址寻址	EA=[基址寄存器]+[变址寄存器]	16 位和 32 位寻址
基址加比例变址寻址	EA=[变址寄存器]×比例因子+[基址寄存器]	32 位寻址
带位移的基址加变址寻址	EA=[变址寄存器]+[基址寄存器]+位移量	16 位和 32 位寻址
带位移的基址加比例变址寻址	EA=[变址寄存器]×比例因子+[基址寄存器]+位移量	32 位寻址

理解存储器寻址方式时,还要注意如下几点:

(1) 有效地址表达式中的位移量是有符号整数,为常量,并紧跟指令码存放在一起。

(2) 16 位寻址和 32 位寻址时,可用作基址、变址和比例因子、位移量的取值规定有所不同。16 位寻址时,BX、BP 是基址寄存器,SI、DI 是变址寄存器;而在 32 位寻址时,任何 32 位通用寄存器都可用作基址寄存器,除 ESP 外的任何 32 位通用寄存器都可用作变址寄存器。

(3) 在许多情况下,可使用段超越前缀来规定访问的逻辑段。缺省段超越前缀时,按约定的原则访问逻辑段。使用 BP、EBP 或 ESP 作为间址、基址或变址寄存器时,默认的逻辑段为 SS **堆栈段**;使用其他间址、基址或变址寄存器时,约定访问的逻辑段为 DS **数**